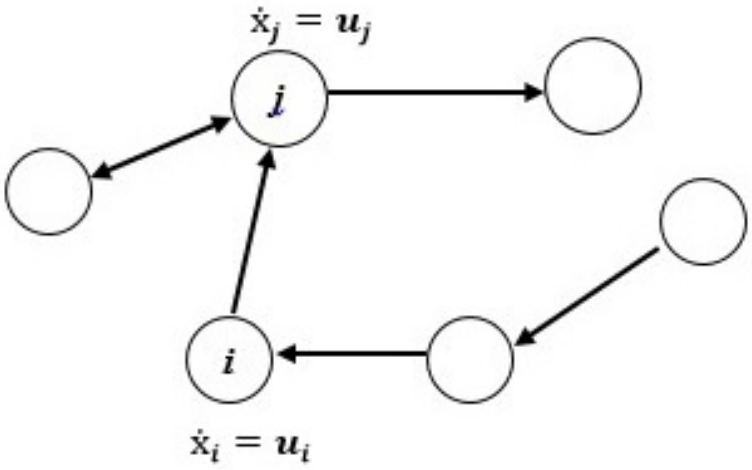
Ricerca Operativa

SIMPLESSO PER LE RETI

A.A. 2022/2023



Andrew Gagliotti**(0000914752)**,
Diego Giuseppetti**(0000988034)**

Indice

1 Introduzione

1.1 Contesto storico

Successivamente alla seconda guerra mondiale, l'impatto della descrizione matematica del mondo reale divenne di importanza vitale. Storicamente era stata già dimostrata l'importanza della statistica e della probabilità al servizio dello spionaggio e della valutazione strategica dei campi di battaglia.
Oggi giorno, la Ricerca Operativa ha l'obiettivo di risolvere problemi di ottimizzazione all'interno delle industrie, andando a creare e migliorare gli algoritmi e i processi di produzione innovativi, con annesso risparmio di tempo e risorse.
In particolare, lo scopo di questo documento è quello di analizzare una delle applicazioni possibili dell'invenzione fatta da *George B. Dantzig* nel 1947, ovvero l'algoritmo del simplesso, approfondendo la sua applicazione alle reti di telecomunicazione.

1.2 Il simplesso e I grafi

L'algoritmo del simplesso è uno fra i metodi risolutivi dei problemi di programmazione lineare, ovvero problemi legati alla minimizzazione (o massimizzazione) di una funzione obiettivo lineare in presenza di vincoli anch'essi lineari. I vincoli lineari del problema sono equazioni o disequazioni introdotti per limitare i valori che la funzione obiettivo può assumere, facenti parte quindi di una *regione ammissibile*.
L'algoritmo del simplesso, nelle sue iterazioni, assicura che venga mantenuta sempre l'ammissibilità della soluzione e che vengano rispettate le condizioni di complementarietà: in breve, un problema di minimo è riscrivibile come problema di massimo (e viceversa) con opportune trasformazioni in base alla validità delle loro soluzioni ottime.
Ecco la formulazione teorica dei problema di minimo e di massimo:

$$\begin{gathered}\text{Min z} = cx \\ \text{s.t. } Ax \geq b \\ x \geq 0\end{gathered}$$

Chiameremo questa formulazione *simplesso primale*, in cui x è il vettore delle variabili decisionali, c è ilvettore dei coefficienti di costo per ciascuna variabile, b è il termine noto per ciascun vincolo, A è la matrice dei coefficienti delle variabili x di ciascun vincolo e z è il valore della funzione obiettivo da minimizzare.

$$\begin{gathered}\text{Max z} = wb \\ \text{s.t. } wA \leq c \\ w \geq 0\end{gathered}$$

Chiameremo questa formulazione *simplesso duale*, in cui w è il vettore delle variabili decisionali, b è il vettore dei pesi associati a ciascuna variabile, c è il termine noto per ciascun vincolo, A è la matrice dei coefficienti delle variabili w di ciascun vincolo e z è il valore della funzione obiettivo da massimizzare.

La teoria dei grafi si appresta a descrivere correttamente quella che è la struttura e il desing di una rete generica: siamo in presenza di un grafo orientato, o *digrafo* i cui archi sono caratterizzati da un costo e da una quantità di flusso massimo trasportabile, mentre i nodi possiedono un potenziale e una disponibilità che indica quanto flusso deve complessivamente passare o è richiesto all'interno. Più dettagliatamente, per la descrizione delle reti si sfruttano gli *spanning tree*, alberi di copertura che simulano un problema di programmazione lineare in cui si vuole risolvere il problema del *flusso di costo minimo*. Non solo, quello che si cerca di ottenere in fase di design, strutturazione e prima accensione di una rete è un insieme di ottimizzazioni

che si possono riassumere con la trasmissione di più dati possibile (quindi un problema di *flusso massimo*), al minor costo possibile e minimzzando il più possibile l'eventuale presenza di errori.
Un generico digrafo ha la seguente formulazione:

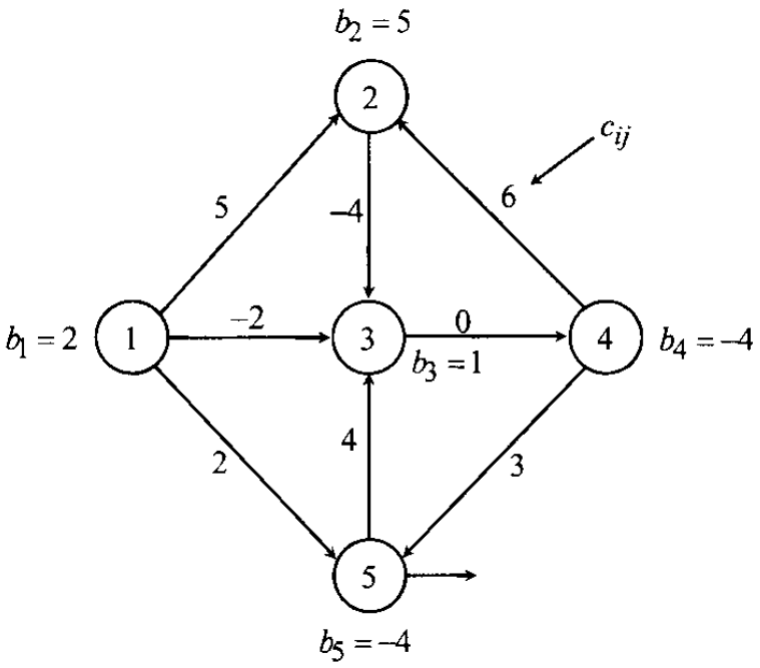


Figura 1: Nella precedente figura[1] è rappresentato uno scenario possibile del Network Flow problem: dato un grafo che simula una rete, si vuole risolvere tramite l'algoritmo del simplesso un problema di flusso di costo minimo in cui ogni arco possiede un costo di trasmissione del flusso, c, ogni nodo possiede una quantità di flusso, b.

1.3 Problemi sulle reti

Minimum cost flow (*MCF*): data una rete con archi a capacità limitata, e un costo per ogni unità di flusso distribuita su uno o più nodi della rete, si vogliono trovare i percorsi che permettano di trasportare queste unita verso nodi richiedenti, pagando il meno possibile. Questo genere di problema è riscontrabile anche al di fuori del settore delle reti, basti pensare alla distribuzione di materie prime,

di prodotti dalle fabbriche e ai problemi di circolazione attraverso i centri urbani delle città.

Sia $G = (N, A)$ una rete con archi orientati, N inisieme di n nodi, A insime e di m archi orientati, per ogni arco $(i, j) \in A$ viene definito un costo c_{ij}, una capacita' massima u_{ij} e un minimo flusso che deve passare per l'arco l_{ij}. Associamo ad ogni nodo $i \epsilon N$ in numero intero $b(i)$. Se $b(i) > 0$ si tratta di un nodo sorgente, se $b(i) < 0$, i e' un nodo destinazione, mentre se $b(i) = 0$ si tratta di un nodo intermedio in cui vige la *conservazione del flusso.* Per ogni arco $(i, j) \epsilon A$ viene asseganta una variabile decisionale x_{ij}.

Formulazione matematica

$$\text{Min} \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \text{ per ogni } i \in N$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \text{ per ogni } (i, j) \in A$$

$$\text{dove} \sum_{i=1}^{n} b(i) = 0$$

In forma matriciale questo problema viene descritto come segue [1.3]:

$$\begin{gathered} \text{Min } cx \\ \text{s.t. } \nu N x = b \\ l \leq x \leq u \end{gathered}$$

Shortest path (*SP*) è il problema sulle reti piu' semplice da risolvere: si vuole trovare un percorso più economico possibile da percorrere nella rete partendo da un nodo s verso un nodo t assumendo che ogni arco $(i, j) \in A$ abbia un costo c_{ij}. Alcune delle applicazioni prevedono di trovare il percorso piu' corto, o che attraversi nel minor tempo possibile, o che abbia il massimo della affidabilita'. Imponendo

$b(s) = 1$, $b(t) = -1$ e $b(i) = 0$ per tutti gli altri nodi nel problema, la soluzione ci permettern il percorso piu' corto possibile.

Maximum flow (*MF*) è il problema complementare di *SP*: mentre questo si concentra sui costi e ignora i limiti di capacita', *MF* ignora i costi per ottimizzare il flusso. Data una rete con dei limiti di capacità sugli archi, si vuole trovare il percorso che permetta di trasportare la maggiore quantità di flusso (*l'informazione*) possibile: se interpretiamo u_{ij} come il massimo flusso possibile del nodo (i, j), *MF* trova il massimo flusso costante tra il nodo s e il nodo t per unita' di tempo. Questo problema è riconducibile a *MCF* se procediamo come segue:

- $b(i) = 0$ per ogni $i \in N$
- $c_{ij} = 0$ per tutti i $(i, j) \in A$
- introduciamo un arco addizionale (t, s)
- costo $c_{ts} = -1$
- limite al flusso $u_{ts} = \infty$

Cosi' facendo *MCF* (*Multi-Commodity Flow* problem) massimizzera' il flusso nell'arco (t, s), ma dato che il flusso su questo arco ha direzione $s \to t$ passando per gli archi in A dove ogni nodo ha $b(i) = 0$, la soluzione di *MCF* massimizzera' il flusso da s verso t. La soluzione matematica di questi problemi viene ottunuta in modo molto semplice: siccome è prevista l'enumerazione di tutte le possibili soluzioni, si sceglie di usare degli algoritmi che permettano di eseguire i calcoli in tempo computazionale piccolo o al massimo accettabile.

2 Il modello matematico

2.1 Il simplesso sulle reti

Dovendo risolvere un problema di flusso di costo minimo si può implementare un metodo del simplesso "ad hoc, più efficiente del metodo standard"[2]. Abbiamo già visto nel paragrafo 1.3 come poter scrivere il problema del flusso di costo minimo in forma matriciale; da questa formulazione posso suddividere le colonne della matrice A in 3 sottinsiemi:

- **B**: variabili base, ovvero quelle per cui vale $0 \leq x_{ij} \leq u_{ij}$;
- **L**: variabili non base tali che $x_{ij} = 0$. Queste sono le variabili il cui valore rappresenta il lower bound di capacità;
- **U**: variabili non base tali che $x_{ij} = u_{ij}$. Queste sono le variabili il cui valore rappresenta l'upper bound di capacità;

Ecco lo pseudocodice dell'algoritmo del simplesso[2]:

Algorithm 1 Simplesso sulle reti

```
Determina una struttura (T, L, U) ammissibile;
Calcola il flusso x corrispondente a (T, L, U);
Calcola i potenziali y corrispondenti a (T, L, U);
while esiste arco non ammissibile do
    aggiungi l'arco (k,l) a T e determina l'arco (p, q) uscente
    Aggiorna la struttura (T, L, U);
    Aggiorna i potenziali y e il flusso x;
end while
```

Questo algoritmo deve essere in grado di determinare una nuova soluzione base ammissibile x, rispettando i potenziali e le capacità degli archi. Queste operazioni critiche si possono risolvere facilmente adottando delle sostituzioni in avanti o all'indietro su un sistema lineare, dopo aver ridotto la matrice dei coefficienti B a una matrice

triangolare inferiore. I passi da seguire sono descritti nei paragrafi successivi.

2.1.1 Determinare una base ammissibile di partenza

Dati n nodi, so che questi rappresentano esattamente n vincoli decisonali. Per essere certi di avere una matrice di base B di colonne linearmente indipendenti, devono essere soddisfatti almeno *n - 1* vincoli (l'ultimo sarebbe automaticamente dimostrato), tenendo a mente che le variabili di base rispettato la relazione $0 \leq x_{ij} \leq u_{ij}$: non eccedono il flusso, eslcudono l'esistenza di un flusso negativo e assumono valori tra U e L esclusi.
La variabile *n-esima* viene data per assunta e inizializzata con valore nullo: sapendo ciò, tutte le altre *n - 1* variabili base vanno a formare uno spanning tree rappresentante una base ammissibile da sottoporre alle condizioni di ottimalità. Vediamo tutto con un esempio per facilitarci le cose:

$$\text{Min z} = 10x_{12} + 12x_{13} + 6x_{14} + 6x_{25} + 2x_{32} + 3x_{34} + 7x_{35} + 3x_{45}$$

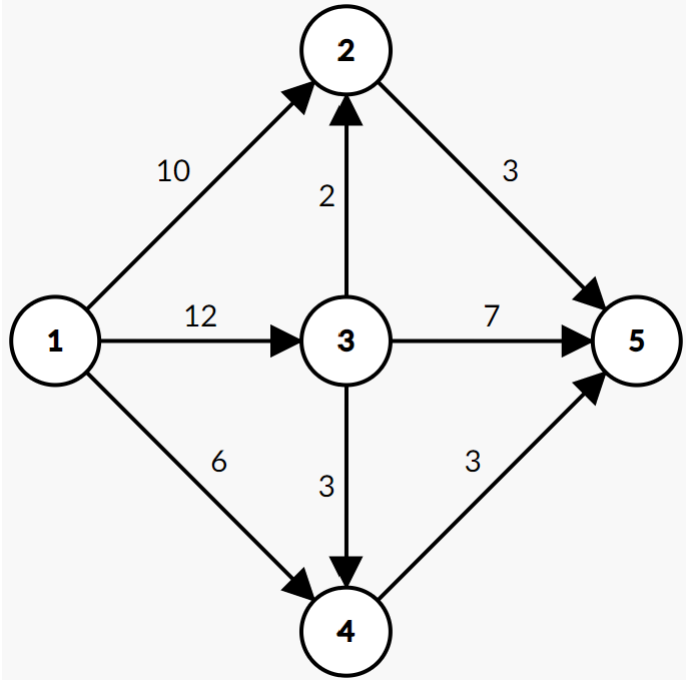
s.t.

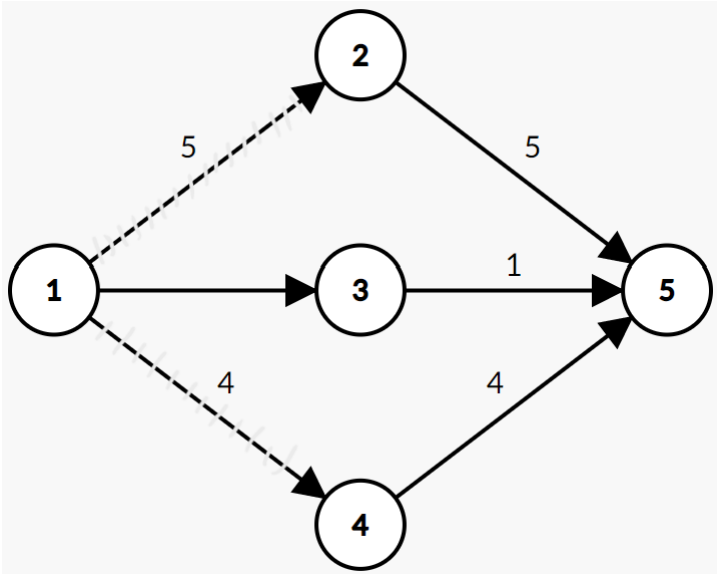
$$\text{x}_{12} + x_{13} + x_{14} = 10$$
$$\text{x}_{25} - x_{12} - x_{32} = 0$$
$$\text{x}_{32} + x_{34} + x_{35} - x_{13} = 0$$
$$\text{x}_{45} - x_{14} - x_{34} = 0$$
$$\text{-}x_{25} - x_{35} - x_{45} = -10$$

a cui sono associati i seguenti vincoli di capacità:

$$0 \leq x_{12} \leq 5; 0 \leq x_{13} \leq 3; 0 \leq x_{14} \leq 4; 0 \leq x_{25} \leq 6$$
$$0 \leq x_{32} \leq 2; 0 \leq x_{34} \leq 5; 0 \leq x_{35} \leq 5; 0 \leq x_{45} \leq 6$$

e i cui grafi (prima il problema di partenza e poi quello rappresentante la prima base ammissibile trovata) sono:





Le variabili non in base sono rappresentate con un arco tratteggiato perchè per quel cammino l'arco è saturato: solo per le variabili in base vale in vincolo

$$\mathrm{L} < x < U.$$

2.1.2 Valutazione della riga 0

Come previsto nell'algoritmo del simplesso si parte osservando la riga 0 (la funzione obiettivo) valutando l'ottimalità della soluzione. Se la soluzione non è ottima, bisogna migliorare il valore attuale della funzione obiettivo di una quantità definita dal vettore $w = c_{bv}B^{-1}$ con

- c_{bv} i valori dei potenziali del flusso,
- B^{-1} matrice inversa della base ammissibile,

da usare sulla riga 0 tramite la seguente sostituzione:

$$c_{ij} = wa_{ij} - c_{ij}$$

che per le variabili in base, con un po' di calcoli dati come dimostrazione al lettore, si adatta in:

$$y_i - y_j = c_{ij}$$

mentre per le variabili non in base diventa:

$$\tilde{c_{ij}} = y_i - y_j - c_{ij}$$

Calcolati i costi, bisogna controllare le condizioni di ottimalità e capire se una delle variabili in base deve uscire in favore di una non in base. Per fare ciò, devo vedere quale è il suo costo a seconda che questa si trovi al lower bound o all'upper bound:

- se una variabile non in base è al lower bound, la base è ottima se i costi sono ≤ 0.
- se una variabile non in base è all'upper bound, la base è ottima se i costi sono ≥ 0.

Dovendo risolvere il problema del flusso di costo minimo, l'algoritmo sfrutta il costo di trasferimento di ogni unità di flusso come condizione

di ottimizzazione e per capire quali variabili devono essere o meno in base. Seguendo l'esempio:

$$y_1 = 0, y_1 - y_3 = 12, y_2 - y_5 = 6, y_3 - y_5 = 7, y_4 - y_5 = 3$$

da cui:

$$y_1 = 0, y_2 = -13, y_3 = -12, y_4 = -16, y_5 = -19$$

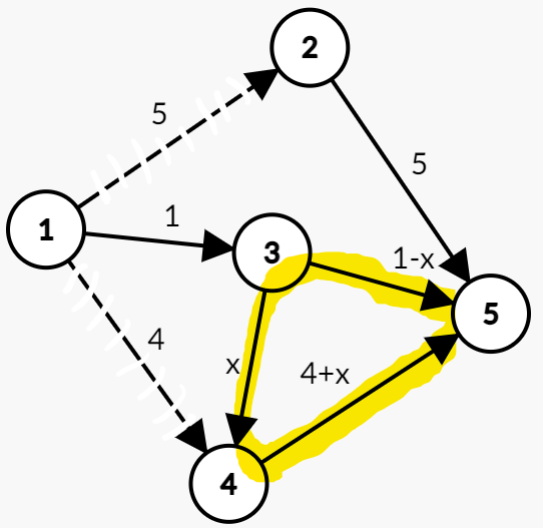
e per quelle non in base vedo:

$$\tilde{c_{12}} = y_1 - y_2 - c_{12} = 3$$
$$\tilde{c_{14}} = y_1 - y_4 - c_{14} = 10$$
$$\tilde{c_{32}} = y_3 - y_2 - c_{32} = -1$$
$$\tilde{c_{34}} = y_3 - y_4 - c_{34} = 1$$

Lo step 2 termina quando si è di fronte alla soluzione ottima oppure quando una variabile non base viola le condizioni di ottimalità. Seguendo la variabile x_{34} che nel nostro esempio viola il vincolo di validità, bisogna proseguire con lo step successivo.

2.1.3 Risoluzione dei cicli

Individuare una nuova variabile candidata ad entrare in base causa la generazione di un ciclo tra tre archi, problema da risolvere cercando di ridirezionare x unità di flusso attraverso il nuovo arco entrante. Per determinare quale variabile è uscente dalla base, basti analizzare quale, fra gli archi coinvolti nel ciclo, non soddisfi più le condizioni di ottimalità.
Fatto ciò, l'arco uscente viene considerato nullo e il nuovo spanning tree dovrà essere valutato per l'ammissibilità.
Nel nostro esempio abbiamo un ciclo con i tre archi evidenziati:



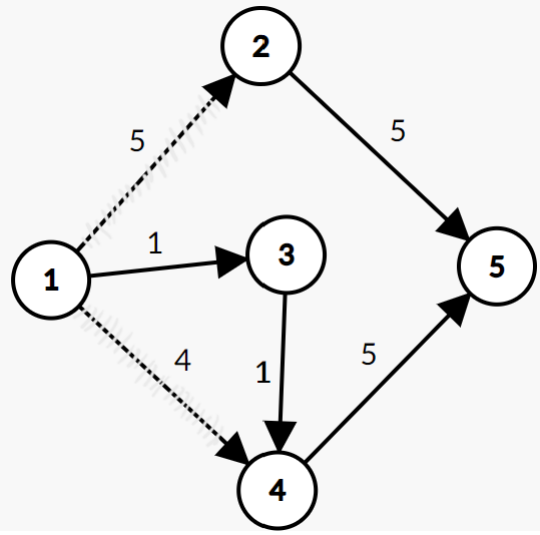
Per l'esempio, è necessario ridirezionale il flusso nell'arco x_{34} sottraendolo a quello in x_{35}. In questo modo l'arco x_{35} è la variabile uscente dalla base che elimina il ciclo, ottenendo un nuovo spanning tree pronto per essere valutato per l'ammissibilità.
Il valore x può essere al massimo 1 per la conservazione del flusso.

2.1.4 Determinare una nuova base ammissibile

A questo punto, bisogna valutare il nuovo spanning per verificare che la nuova base trovata sia ammissibile, per poi ritornare allo Step 2 [2.1.2] per vedere se è stata raggiunta la soluzione ottima.

2.1.5 La soluzione ottima del nostro esempio

Iterando per una seconda volta, l'esempio giunge alla seguente soluzione ottima:



le cui variabili valgono:

$$x_{12} = 5, x_{14} = 4$$
$$x_{32} = x_{35} = 0$$
$$x_{13} = 1, x_{34} = 1, x_{25} = 5, x_{45} = 5$$

e se vado a sostituire questi valori nella funzione obiettivo ottengo:

$$\text{Min z} = 10x_{12} + 12x_{13} + 6x_{14} + 6x_{25} + 2x_{32} + 3x_{34} + 7x_{35} + 3x_{45}$$
$$\text{Min z} = 10 \cdot 5 + 12 \cdot 1 + 6 \cdot 4 + 6 \cdot 5 + 2 \cdot 0 + 3 \cdot 1 + 7 \cdot 0 + 3 \cdot 5$$
$$\text{Min z} = 134$$

Resta poi da verificare la veridicità dei dati anche al calcolatore.

3 Applicazione

3.1 Implementazione in python

La libreria networkx[3] fornisce tutta una serie di funzioni e strutture dati al servizio dell'analisi di grafi, digrafi e multigrafi, con annessi algoritmi e implementazioni al servizio di progettisti delle reti. Essendo l'obiettivo dell'elaborato dimostrare di aver appreso le conoscenze matematiche dell'algoritmo del simplesso sulle reti, per verificare i calcoli matematici al calcolatore abbiamo creato un grafo e dopo averne definito tutte le caratteristiche abbiamo richiamato la funzione di libreria *network-simplex(graph: Graph)*, che restituisce il risultato del problema del flusso di costo minimo come tupla, contenente:

- il cammino che risolve il problema di flusso di costo minimo.
- il risultato della funzione obiettivo: quanto minimo bisogna spendere per far transitare il flusso dalla sorgente alla destinazione.

Creato il grafo e richiamata la funzione:

```
# Initializing a new graph
G = nx.DiGraph()

# Adding nodes
G.add_node("1", demand=-10)
G.add_node("2", demand=0)
G.add_node("3", demand=0)
G.add_node("4", demand=0)
G.add_node("5", demand=10)

# Adding edges
G.add_edge("1", "2", weight=10, capacity=5)
G.add_edge("1", "3", weight=12, capacity=3)
G.add_edge("1", "4", weight=6, capacity=4)
G.add_edge("3", "2", weight=2, capacity=2)
G.add_edge("3", "4", weight=3, capacity=5)
G.add_edge("3", "5", weight=7, capacity=5)
G.add_edge("2", "5", weight=6, capacity=6)
G.add_edge("4", "5", weight=3, capacity=6)
flowCost, flowDict = nx.network_simplex(G)
```

L'output sarà:

```
Flusso:  {'1': {'2': 5, '3': 1, '4': 4}, '2': {'5': 5},
'3': {'2': 0, '4': 1, '5': 0}, '4': {'5': 5}, '5': {}}
Min z =  134
```

Il risultato della funzione obiettivo e il cammino nel grafo sono stati provati al calcolatore.

Riferimenti bibliografici

[1] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[2] Marco Antonio Boschetti. *Network Simplex Method*. Marco Antonio Boschetti, 2023.

[3] NetworkX developers. *https://networkx.org/*. NetworkX developers, 2013-2014.